

MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models

Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, Gianluca Stringhini.

NDSS 2017, 28-02-2017



Motivation: Android & Malware

- **Android market share is growing**
 - In 2016, 85% of smartphone sales
- **At the same pace the interest by cybercriminals is growing**
 - Bypassing two-factor authentication
 - Stealing sensitive information, etc.

Motivations: Current Defenses

- **Can't use complex on-device operations**
 - Limited battery and memory resources
- **Google's centralized analysis**
 - Previous work shown a few incidents
 - Users buy apps from third party markets
- **Lots of research in the field! However**
 - Permission-based models prone to false positive
 - Relying on API calls frequently used by malware needs constant, costly retraining

Motivations: Our Idea

Intuition: malware uses calls for different actions and in different order than benign apps

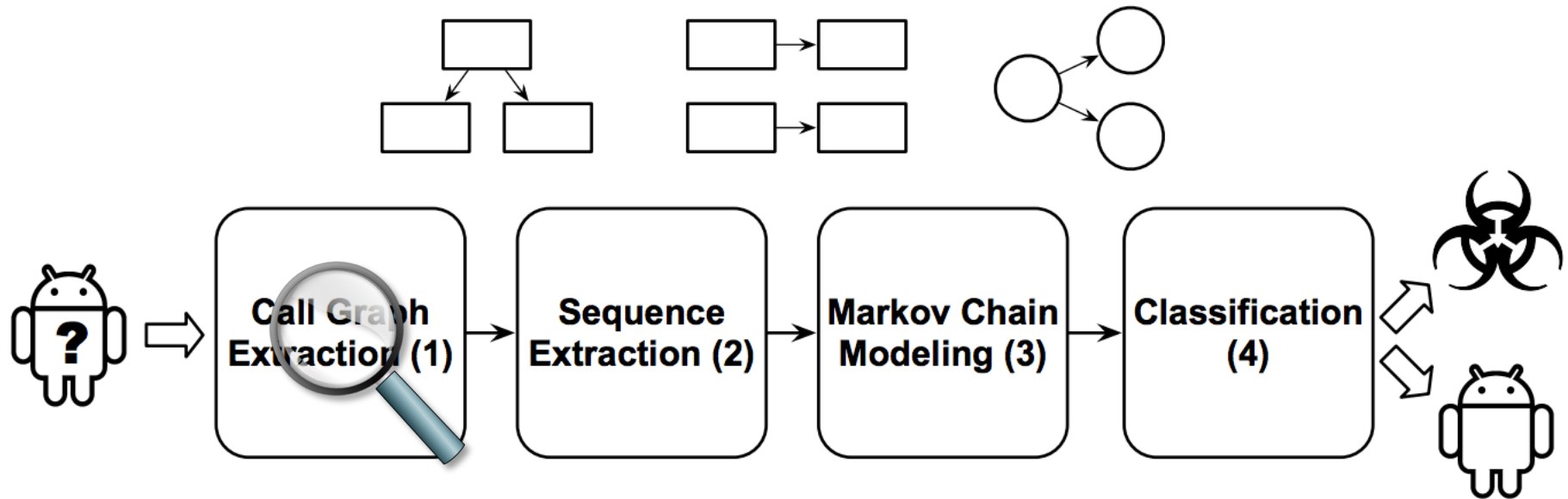
- E.g. `android.media.MediaRecorder` used by any app with permission to record audio
- Only using it after calls to `getRunningTasks()`, which allows to record conversations, may suggest maliciousness

Rely on the **sequence of **abstracted** calls**

1. Sequence captures the behavioral model
2. Abstraction provides resilience to API changes

MaMaDroid

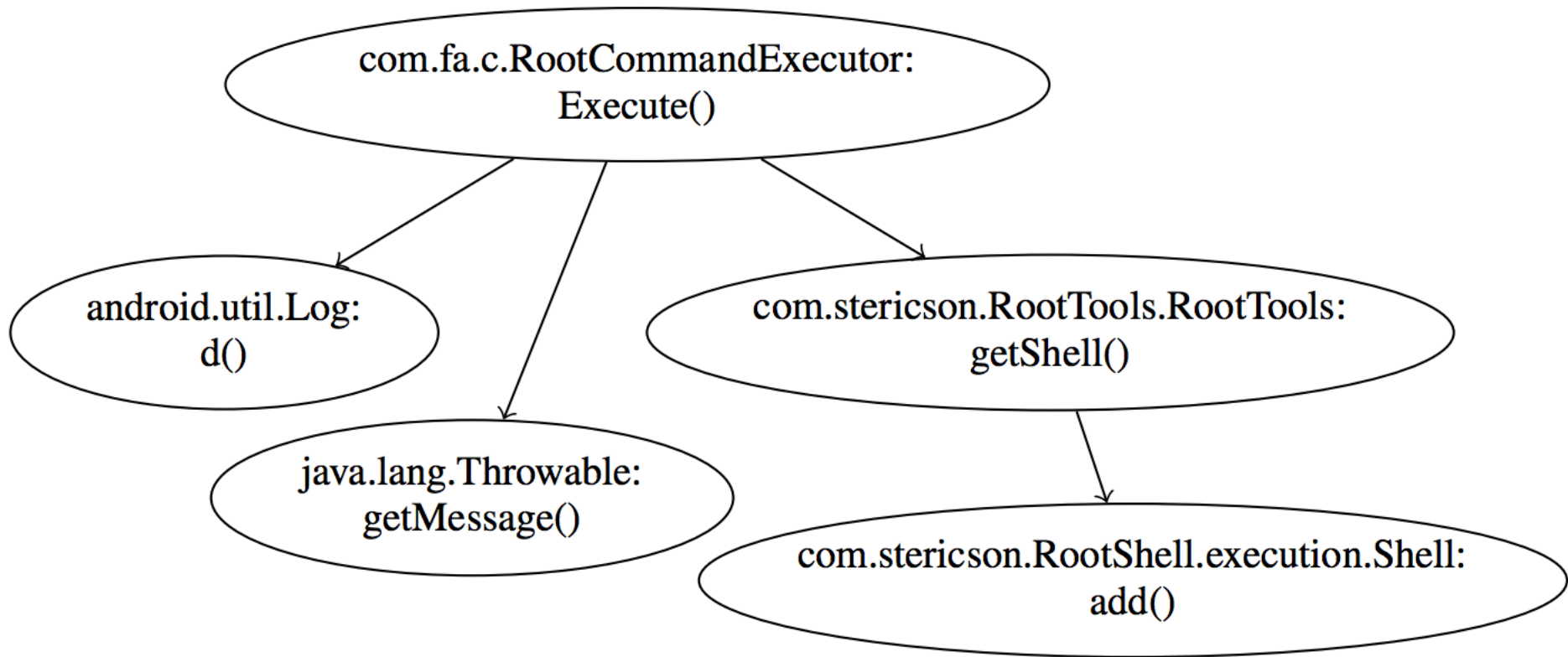
Overview



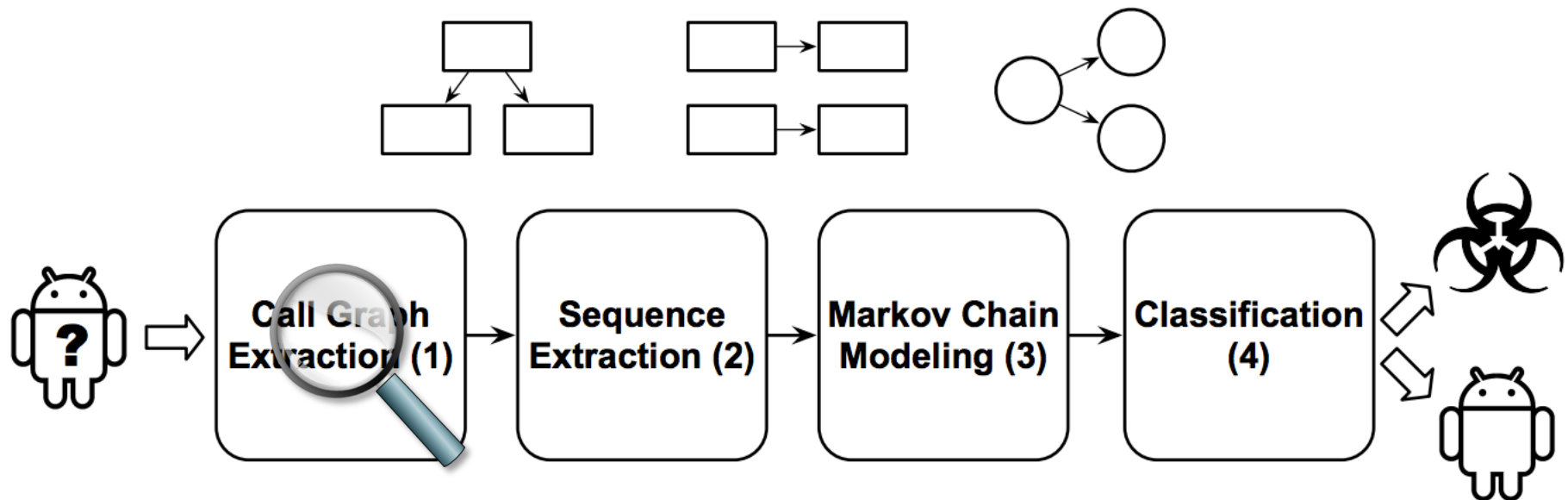
Call Graph Extraction

- **Based on static analysis**
 - Given an apk, extract call graphs
- **Tools**
 - Soot (Java optimization and analysis framework)
 - FlowDroid

Call Graph



Overview



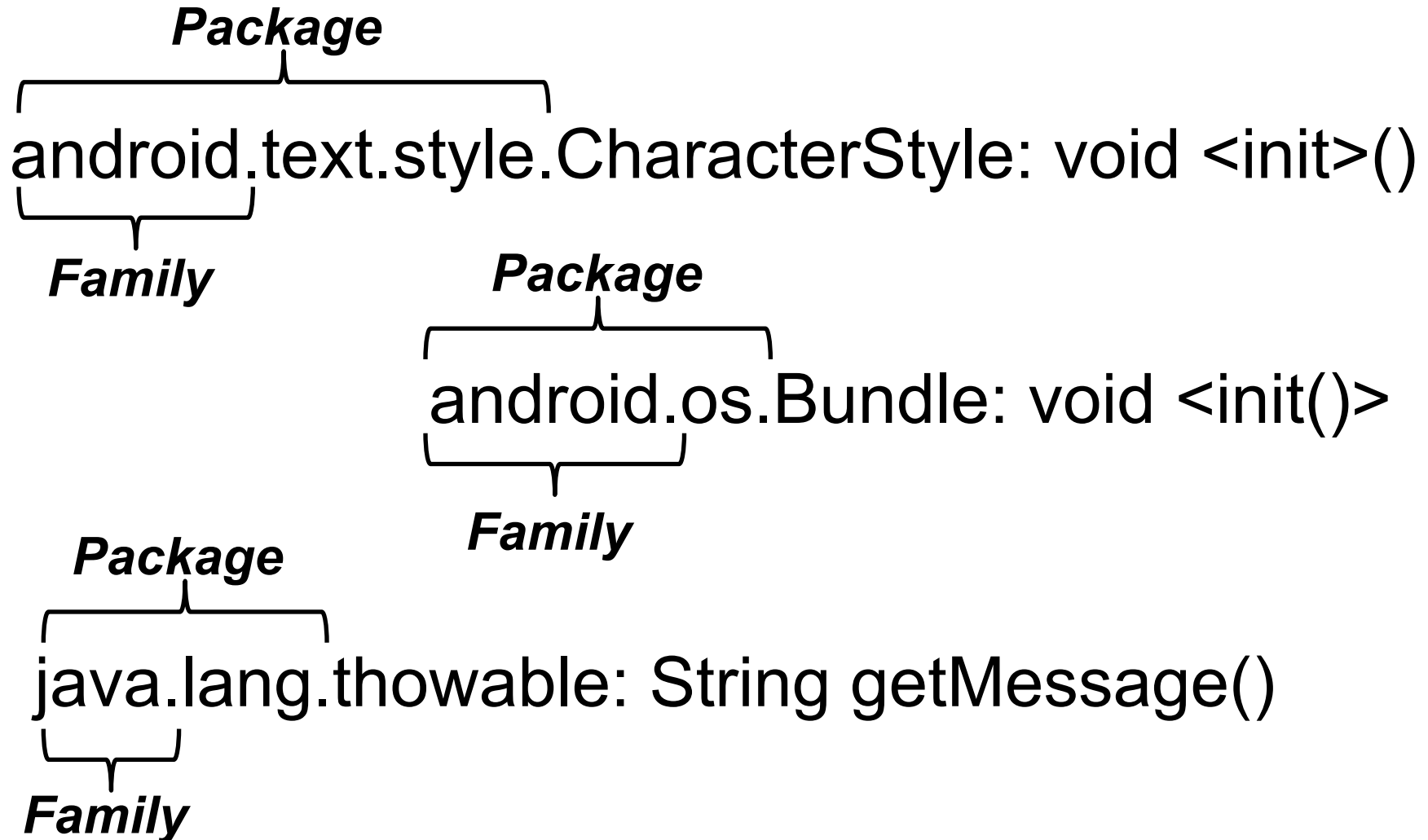
Sequence Extraction

- **Soot gives the call graph from which we extract the sequence of functions that are potentially called by the program, but...**
- **When running example multiple times...**
 - Execute() may be followed by different calls, e.g., getShell() only in try or getShell() + getMessage() in catch

Sequence Extraction

- **We proceed as follows:**
 1. Identify set of entry nodes
 2. Enumerate paths
 3. Output set of all paths as the sequences of API calls
- **But we said we were using abstracted calls!**

Abstraction



Abstraction

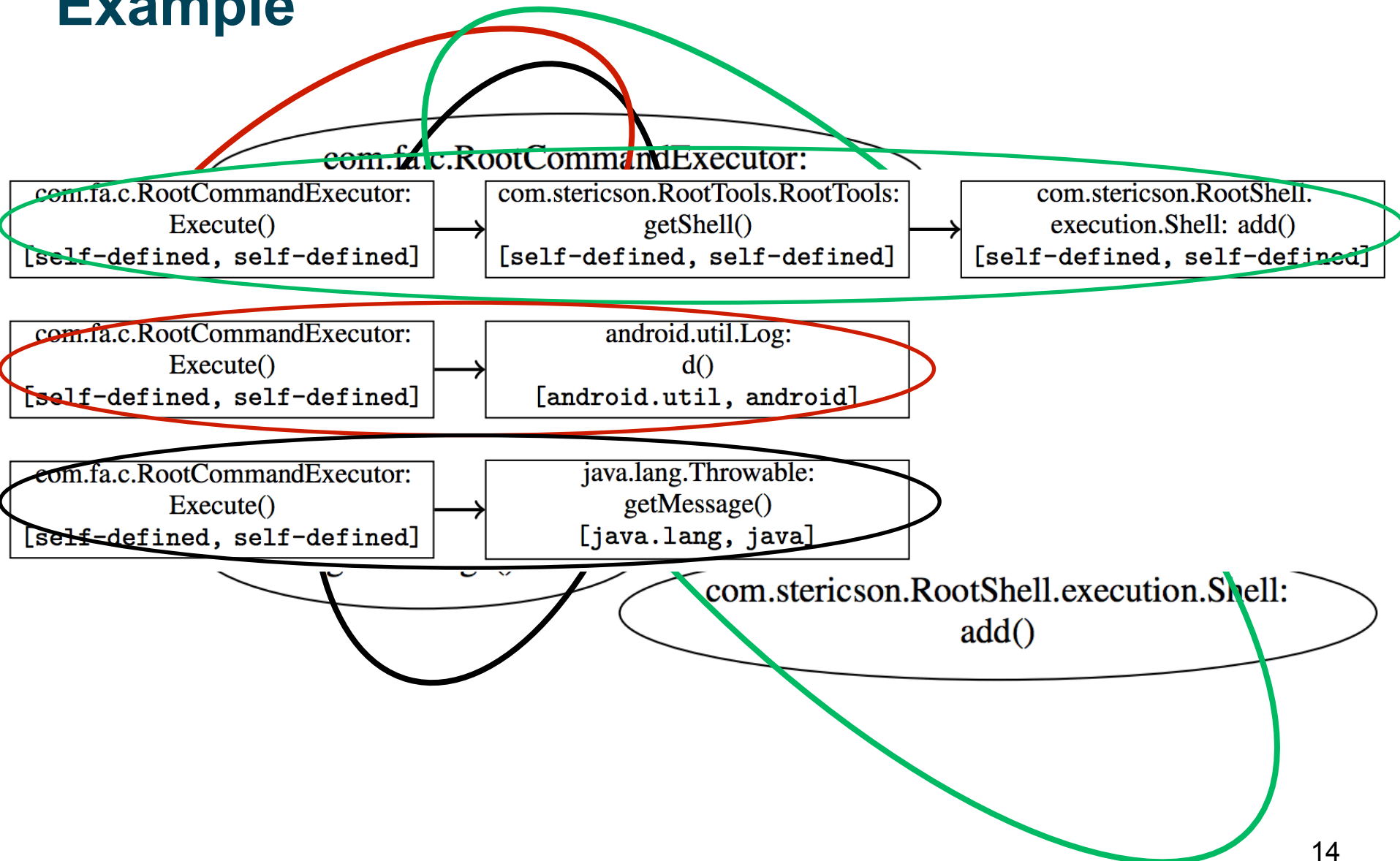
- **Packages**

- Using the list of 243 packages (as of API level 24) + 95 from the Google API
- Packages defined by developers → “self-defined”
- If we can’t tell what its class implements → “obfuscated”

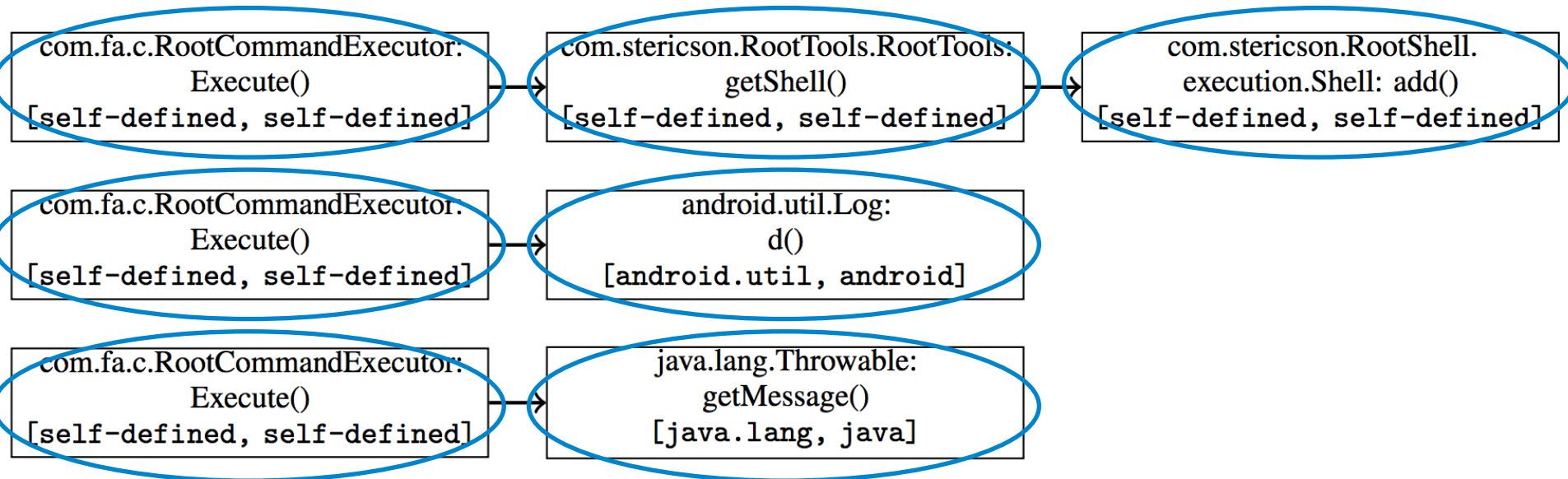
- **Families**

- 9 families: android, google, java, javax, xml, apache, junit, json, dom
- Plus self-defined and obfuscated

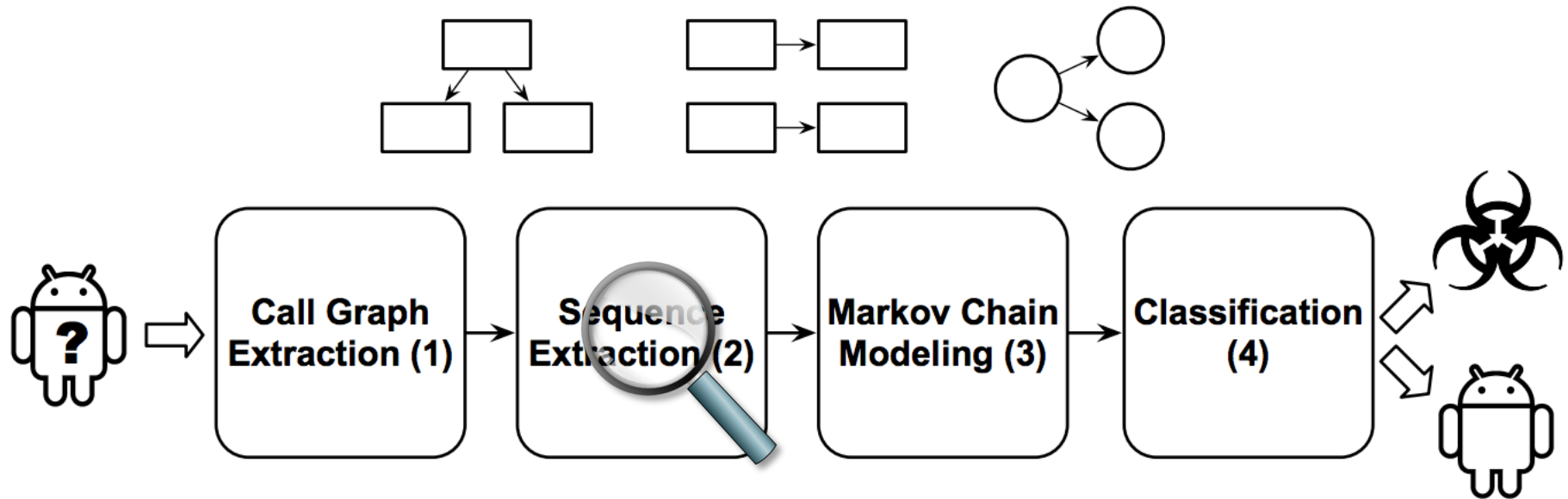
Example



Example



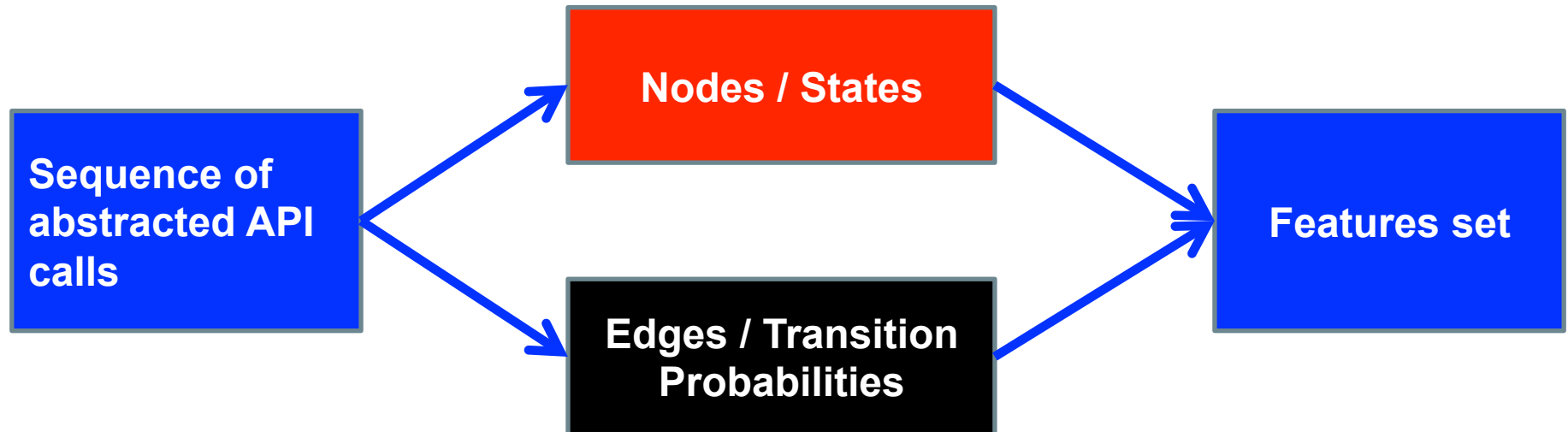
Overview



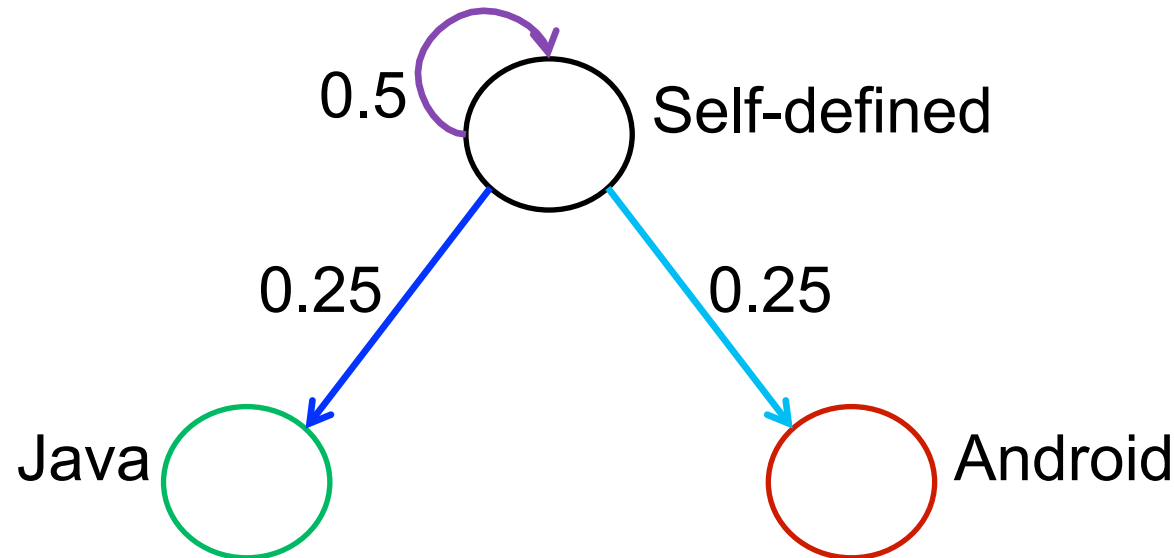
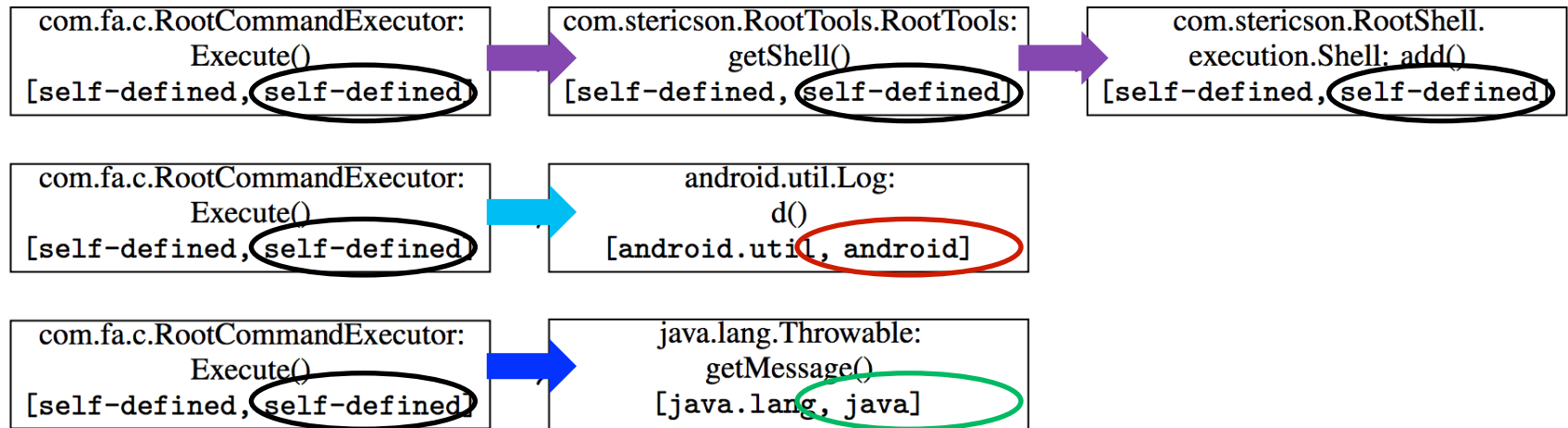
Markov Chain

- Memoryless models
 - Probability of transition from a state to another only depends on the current state
- Represented as a set of nodes
 - Each corresponding to a different state, and a set of edges labeled with the probability of transition.
- Sum of all probabilities associated to all edges from any node is exactly 1

Building the Markov Chains



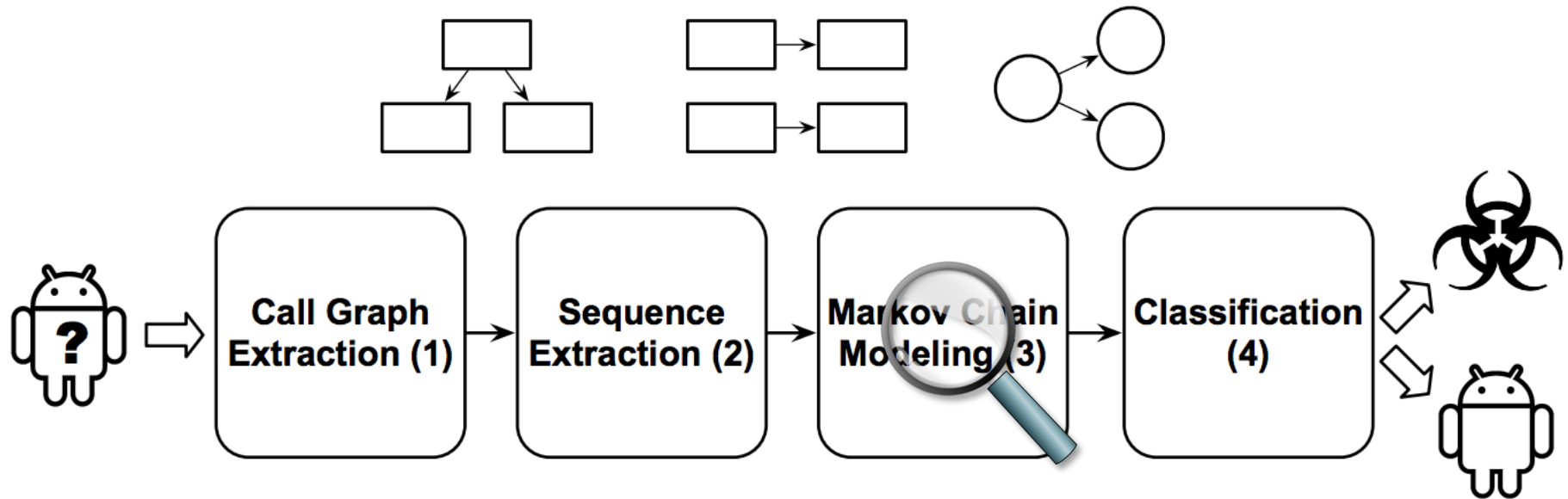
Example



Feature Extraction

- **For each app:**
 - Feature vector = probabilities of transition from one state to another in the Markov chain
 - With families, 11 possible states \rightarrow 121 possible transitions in each chain
 - With packages, 340 states \rightarrow 115,600 transitions

Overview



Classification

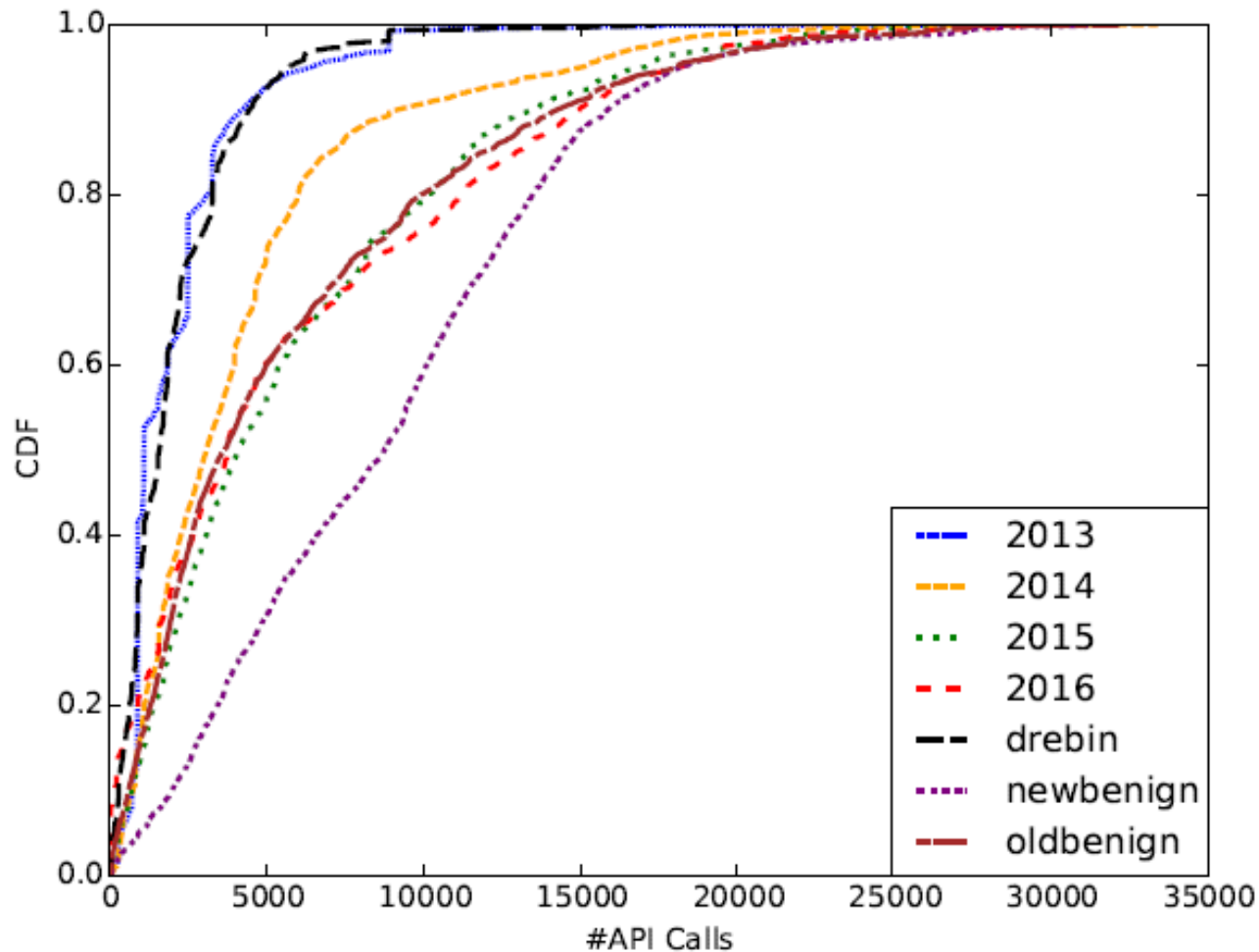
- **Build a classifier using the extracted features**
 - Each app labeled as benign or malware
- **We tested our idea using:**
 - Random Forests
 - 1-NN, 3-NN
 - SVM
- **SVM was less efficient than the other systems**

Dataset

Datasets

Category	Name	Date Range	# Samples	# Samples (API Calls)	# Samples (Call Graph)
Benign	OldBenign	Apr 13 – Nov 13	5,879	5,837	5,572
	NewBenign	Mar 16 – Mar 16	2,568	2,565	2,465
		Total Benign	8,447	8,402	8,037
Malicious	Drebin	Oct 10 – Aug 12	5,560	5,546	5,538
	2013	Jan 13 – Jun 13	6,228	6,146	6,123
	2014	Jun 13 – Mar 14	15,417	14,866	14,827
	2015	Jan 15 – Jun 15	5,314	5,161	4,725
	2016	Jan 16 – May 16	2,974	2,802	2,657
		Total Malicious	35,493	34,521	33,870

How many API calls?

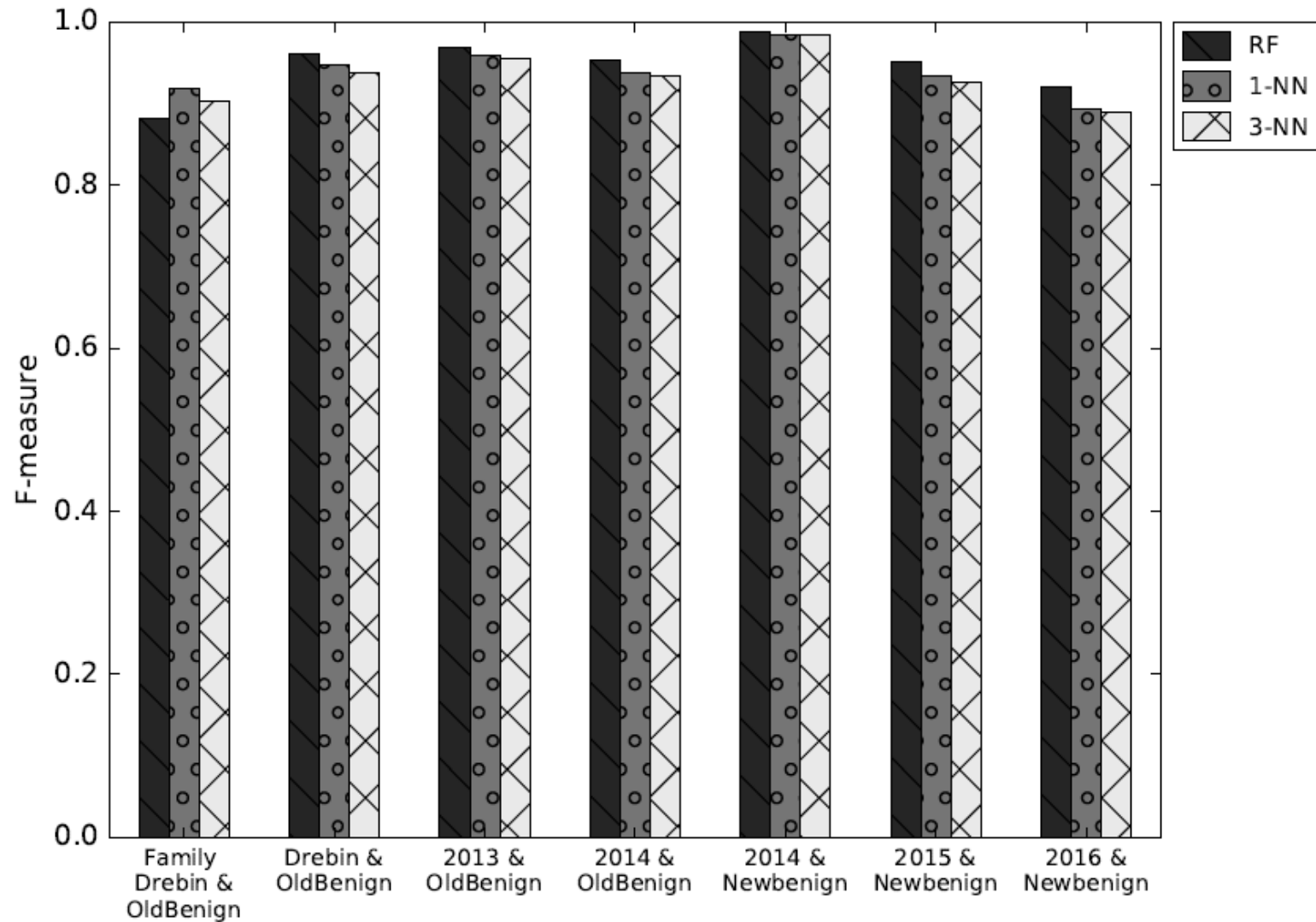


Evaluation

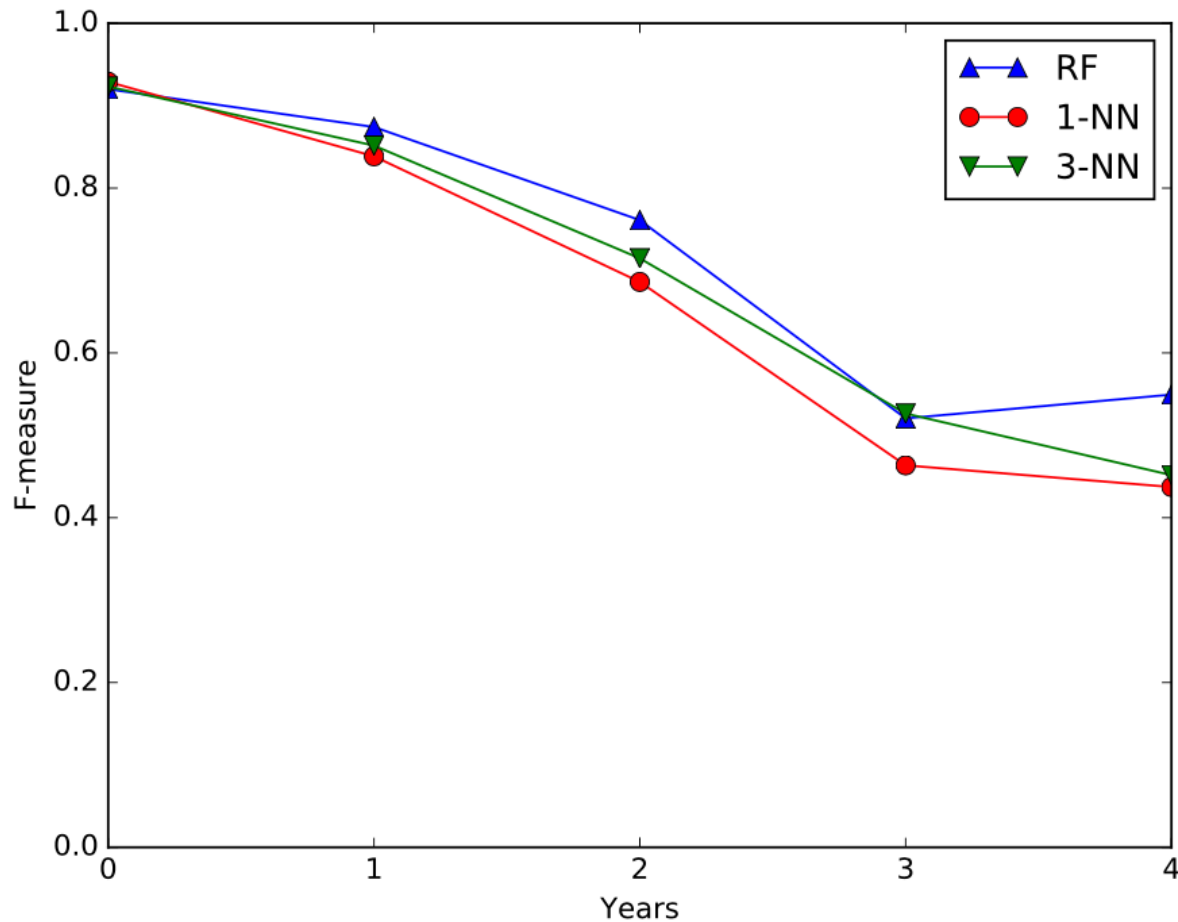
Evaluation

- Accuracy of classification on benign and malicious samples developed around the same time
- Robustness to the evolution of malware as well as of the Android framework (using older datasets for training and newer ones for testing and vice-versa)

Same Year

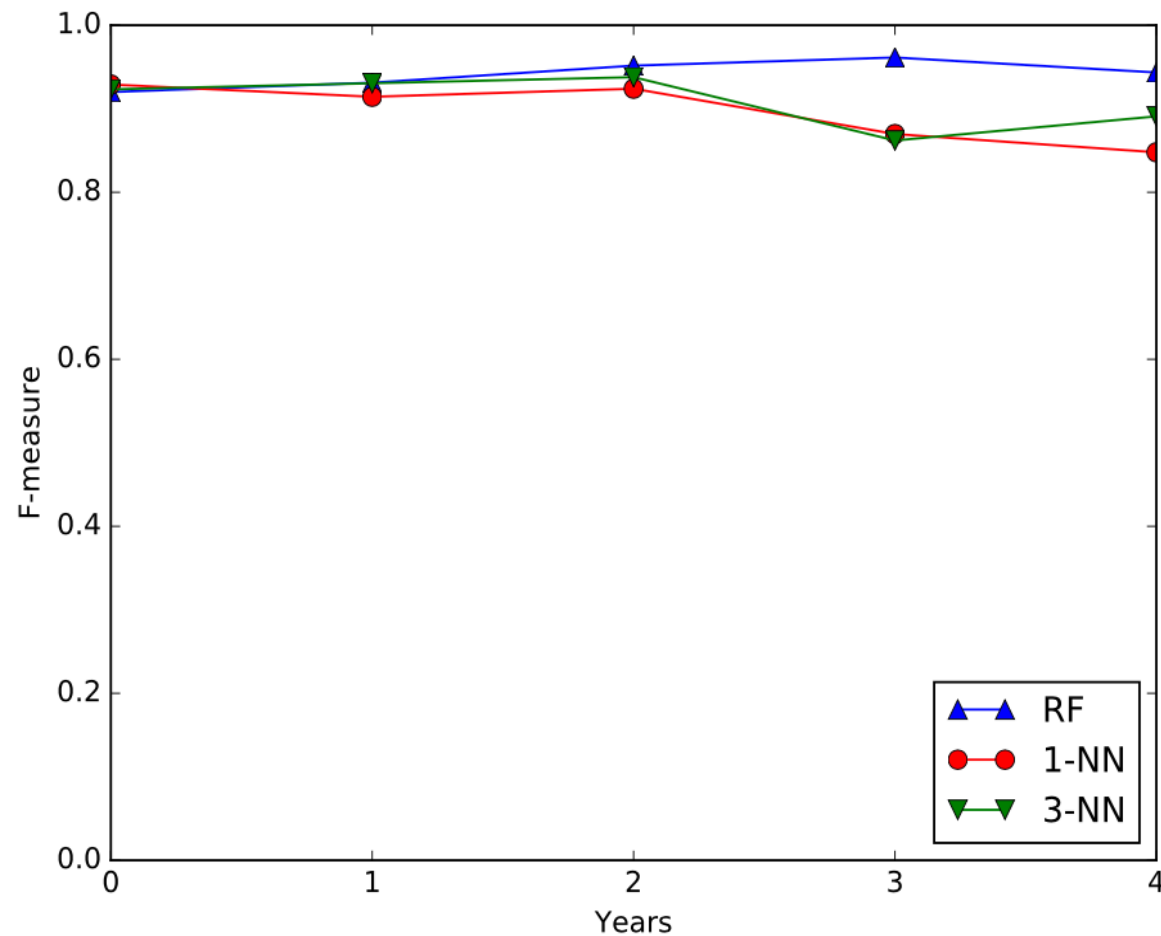


Training on older samples



Families abstraction

Training on newer samples



Families abstraction

MaMaDroid Vs DroidAPIMiner

DroidAPIMiner is the previous work operating detection of malware samples from benign ones based on sequences of API calls.

Tests	DroidAPIMiner	MaMaDroid
Same Year	0.56	0.96
Older samples Training	0.42	0.68
Newer samples Training	0.50	0.96

Discussion and Limitations

Case Studies (2016/newbenign)

- **False Positives (164 samples)**
 - Most of them “dangerous permissions”
 - E.g., SMS permissions not clear why requested
- **False Negatives (114 samples)**
 - Actually not classified as malware by VirusTotal, might be legitimate
 - Most of them adware

Evasion

- **Repackaging benign apps**
 - Difficult to embed malicious code while keeping similar Markov chain, viceversa is also hard
- **Imitating Markov chains**
 - Likely ineffective
- **Obfuscation/Mangling**
 - Still captured by the [obfuscated] abstraction
- **More in the paper...**

Limitations

- Classification is memory hungry
- Soot is buggy, we lose ~4% of the samples
- Limits of static analysis only methods

Future Work

- **Further investigate resilience to evasion**
 - Focus on repackaged malicious apps
 - Injection of API calls to mess with Markov chains
- **Enhancements**
 - Fine-grained abstractions (e.g., class)
 - Seed with dynamic analysis
- **Releasing**
 - MaMaDroid's python code
 - The list of used samples and their hashes
 - Parsed dataset

Conclusions

- We created MaMaDroid, a system for android malware detection
- Static analysis only, based on Markov Chain modeling of sequences of API calls
- Up to 0.99 F-measure in tests, resilient to changes over time



Enrico Mariconti

Thanks for listening